

# Evaluation of Bayesian Optimization applied to Discrete-Event Simulation

Philipp Zmijewski   Nicolas Meseth

University of Applied Sciences Osnabrueck

Oldenburger Landstraße 62

49090 Osnabrueck, Germany

philipp.zmijewski@hs-osnabrueck.de,   n.meseth@hs-osnabrueck.de

Keywords: Simulation, Optimization, Bayesian Optimization, Simio, GPyOpt

**ABSTRACT:***In this paper, we evaluate the application of Bayesian Optimization (BO) to discrete event simulation (DES) models. In a first step, we create a simple model, for which we know the optimal set of parameter values in advance. We implement the model in SimPy, a framework for DES written in Python. We then interpret the simulation model as a black box function subject to optimization. We show that it is possible to find the optimal set of parameter values using the open source library GPyOpt. To enhance our evaluation, we create a second and more complex model. To better handle the complexity of the model, and to add a visual component, we build the second model in Simio, a commercial off-the-shelf simulation modeling tool. To apply BO to a model in Simio, we use the Simio API to write an extension for optimization plug-ins. This extension encapsulates the logic of the BO algorithm, which we deployed as a web service in the cloud. The fact that simulation models are black box functions with regard to their behavior and the influence of their input parameters makes them an apparent candidate for Bayesian Optimization (BO). Simulation models are multivariable and stochastic, and their behavior is to a large extent unpredictable. In particular, we do not know for sure which input parameters to adjust to maximize (or minimize) the model's outcome. In addition, the complex models can take a substantial amount of time to run. Bayesian Optimization is a sequential and self-learning algorithm to optimize black box functions similar to as we find them in simulation models: they contain a set of parameters for which we want to identify the optimal set, they are expensive to evaluate, and they exhibit stochastic noise. BO has proven to efficiently optimize black box functions from various disciplines. Among those, and most notably, it is successfully applied in machine learning algorithms to optimize hyperparameters.*

## 1 Introduction

### 1.1 Situation and Motivation

Finding the optimum in a complex stochastic environment is a difficult task. Simulation is a proven tool to model stochastic processes. However, simulation by itself is no optimization. Therefore, a variety of algorithms have been developed to try and optimize simulation models [1].

In this paper, we examine a set of resource allocation problems and their optimization with Bayesian Optimization (BO). We implement the problems as discrete event simulation (DES) models with the Python simulation package *SimPy* [2] and the graphical simulation modeling tool *Simio* [3]. Both models serve as objective functions to optimize, where a defined output (response) of each model is to be minimized or maximized. The task is to find the best combination of the model's input parameters (controls), taking into consideration a set of constraints imposed on these input parameters.

### 1.2 Related Work

To narrow down the problem domain, we focus on simulation and optimization in a manufacturing context. A lot of research has been done in this context [1][4][5]. Especially, recent applications of self learning algorithms open new possibilities [6][7][8]. Both subfields of this paper, Bayesian Optimization [9][10] and simulation optimization [1][4][5], are well researched. However, we failed to identify relevant research that focuses on the combination of Bayesian Optimization and simulation optimization in the manufacturing domain.

## 2 Bayesian Optimization

### 2.1 Black-Box Functions

The term black-box function implies that the intrinsic behavior of the function is unknown. The only way to obtain information about the function is to through sampling and observation. That is, the function  $f(x)$  is evaluated for different values of it's parameter set  $x \in X$  (Figure 2.1). Moreover, we cannot make any assumption about the function itself, such as its mathematical formula, smoothness or convexity [11]. Therefore, an optimization by derivation is impossible.

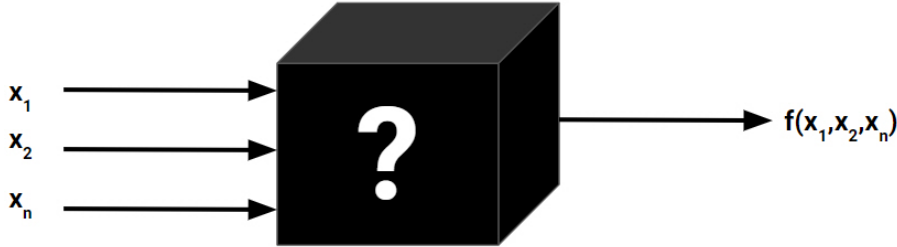


Figure 2.1: Schematic visualization of a black-box function [11].

Given we know the parameter ranges for each parameter  $x_1 \dots x_n$ , along with any given constraints, we could theoretically apply a brute force approach and evaluate every valid combination of parameter values to find the one which minimizes (or maximizes) the objective function. For most problems in practice, the search space is extraordinary large, resulting in unacceptable long computation times. This becomes an even bigger problem when the objective function is costly, i.e. the function requires a substantial amount of time or resources to evaluate.

The aim of black-box optimization is to find the optimal set of parameter values for the objective function with as little total computation as possible [7]. Bayesian Optimization is a technique for finding good (or optimal) parameter values for black-box functions with only few evaluations [10][12].

The black-box definition also applies to simulation models. Simulation models take a number of input parameters (server capacities, queuing and release policies, stock sizing, etc.) and generate one or more responses (throughput time, length of machine queue, server utilization, costs, etc.). Moreover, there is no explicit mathematical description of how the output is generated. The output is rather determined by the emergent behavior of a complex, stochastic model. We can therefore assume simulation models to be black-box functions, for which we can obtain information merely through simulation experiments with different input parameter values, and the observation of the responses.

### 2.2 Noisy and Expensive Functions

Because of stochastic elements such as arrival times, processing times, or failure rates, the responses of a simulation model will be different for each simulation run, even with the same parameter values. This adds uncertainty to the response and makes the objective function *noisy*. To make the optimization even harder, simulations typically take a substantial amount of time to run, which makes them *expensive* or *costly*. Both properties, the noisiness and costliness, can be addressed using Bayesian Optimization.

### 2.3 Sequential Optimization

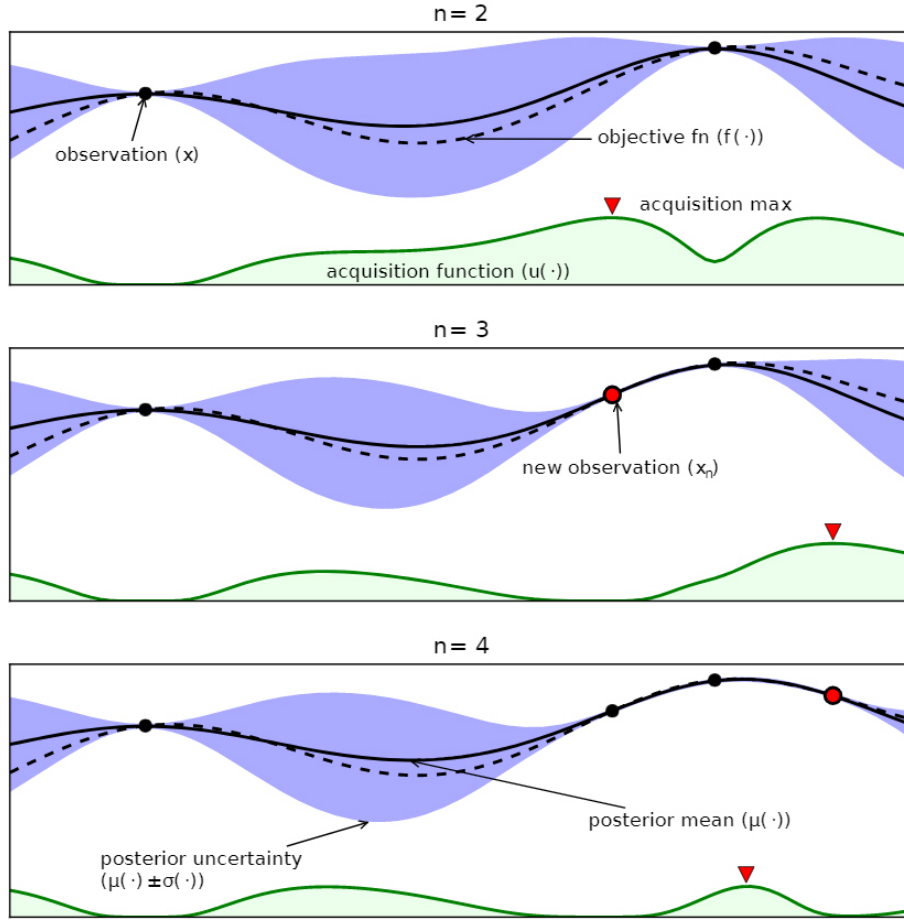


Figure 2.2: An example of using Bayesian optimization including a objective function and an acquisition function with four iterations [8].

Bayesian Optimization is a sequential, self-learning optimization algorithm [9][8]. Since no explicit function is given, BO uses two components: (1) a surrogate model for modeling the objective function, and (2) an acquisition function to decide which parameter values to try with the next evaluation (*here*: simulation run) [9][13] (Figure 2.2).

The objective function is the function to be optimized. In classical BO, we use Gaussian processes to define a prior distribution. With each new observation, we subsequently update the distribution to determine the posterior distribution [13]. That is, with each new observation, we reduce uncertainty about the true function and improve our surrogate model. The acquisition function determines the utility of a new suggested point based on a trade-off of between exploitation (high/low objective function) and exploration (high uncertainty) [9][8]. This technique reduces the number of objective function evaluations and is likely to do well at multi modal functions [8].

In recent years, a number of publications addressed Bayesian Optimization [9][8][14][13]. A breakthrough for BO was the successful application in hyperparameter tuning of machine learning algorithms [8]. The successful application of BO motivated further research from different application fields to solve problems using this method [6][15].

For more detailed information on Bayesian Optimization we refer the reader to [9].

### 3 Applying Bayesian Optimization to Simulation

#### 3.1 Experiment setup

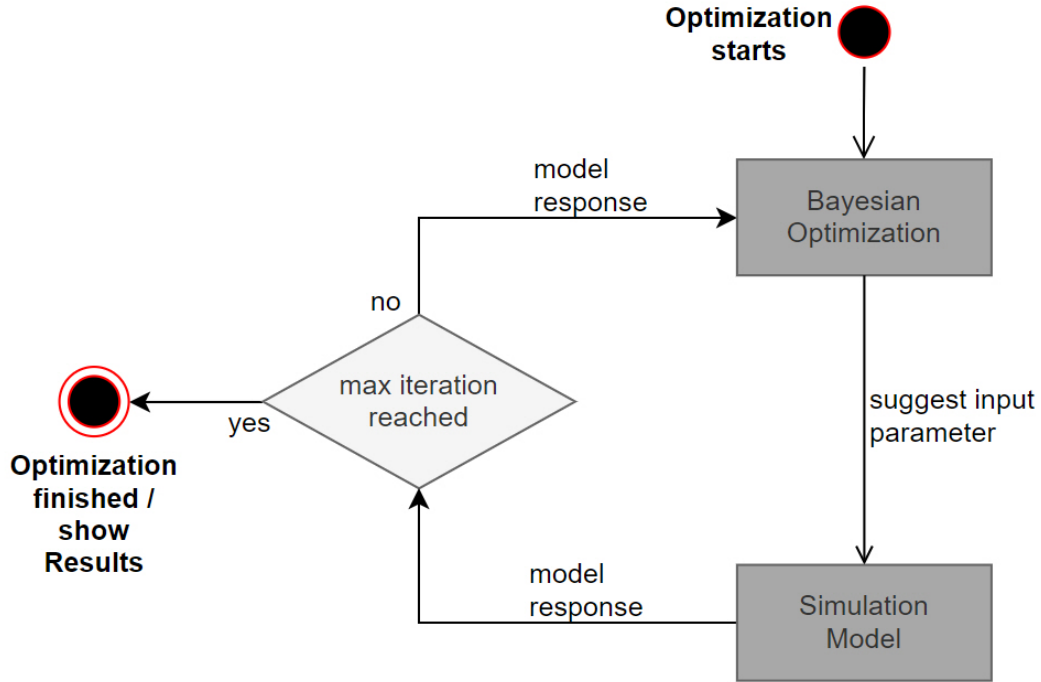


Figure 3.1: Schematic representation of Simulation and Bayesian Optimization.

Our experiment is based on two models, which are synthetic resource allocation problems in a production environment. Both models contain a fixed number of machines, different product types with different work sequences, orders for the different product types, and a limited total number of workers. The aim is to minimize the average time in system (ATS) for all orders by allocating the optimal number workers to each of the machines. The models are:

1. **Deterministic model with 2 machines and 3 products (2M3P-D):** A model with two machines and three products with different sequences. The model is deterministic, all arrivals and processing times are fixed numbers. For this model, we identified the true optimum for the allocation of workers to the two machines using brute force. We implemented this model in SimPy and Simio. In the following, this model is abbreviated 2M3P-D (2 machines, 3 products, deterministic).
2. **Stochastic model with 5 machines and 5 products (5M5P-S):** A model with five machines and five products with different sequences. The model has stochastic elements (processing time and order arrival time). Because of the larger parameter space, we did not obtain the the true optimum through brute force. We implemented this model only in Simio. In the following, this model is called 5M5P-S (5 machines, 5 products, stochastic) (Figure: 3.2).

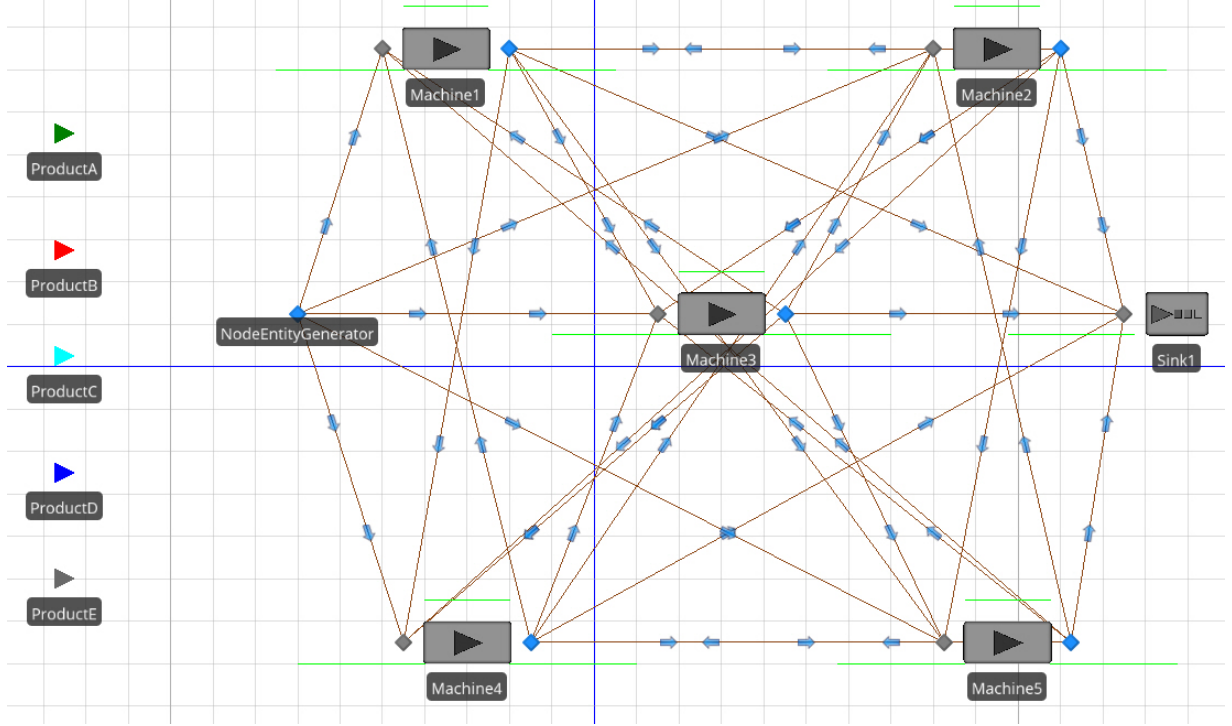


Figure 3.2: Screenshot of the facility view for the 5M5P-S model in Simio.

We employed the Python library GpyOpt [16] to perform the Bayesian Optimization. Simio [3], which we used for models two and three, exposes an API written in .NET to develop so called experiment add-ins. With such an add-in, we can take over the control of experiments and scenario generations and execution in Simio. As GPyOpt is written in Python (and not in a .NET language), we deployed the BO algorithm as a web service. More precisely, we developed and deployed three serverless functions on Google's cloud service. These functions are called by the Simio experiment add-in to perform the optimization loop.

A simplified view of the optimization loop is shown in Figure 3.1. BO starts with 5 random initial points, which are evaluated by the simulation model and the responses sent back to BO. Based on the results, BO suggests the next set of input parameters using the acquisition function. These points are again evaluated by the simulation model, and based on the results, the objective function is updated. This process runs in a loop, until the stopping criteria are met. For our purposes, we defined a single stopping criterion based on a specific number of iterations. Another criterion that is often applied is the minimum distance between two consecutive parameter sets, also called epsilon [16].

In order to understand the experiment, certain terms have to be defined:

1. **Replication:** Number of runs of the simulation model with a fixed set of parameters. For a deterministic model, one run is sufficient. For stochastic models, we have to do multiple runs and build an average of the responses.
2. **Iterations:** Number of points BO evaluates in its attempt to find the optimum. Every iteration has different parameter set. The number of iterations consists of the 5 initial points and the number of subsequently suggested points.
3. **Experiment runs:** Number of full optimization runs using BO with a defined number of iterations. At the end of every run, we check the best value BO has found within the limited number of iterations.

For each experiment run, we determine the number of iterations it took BO to find the optimum, or, if the optimum was not found, the best parameter set and response found by BO within the maximum number of iterations. If the optimum was not found, we calculate the error to know by how far BO was off:

$$Error = x - x^* \quad (3.1)$$

Where  $x$  is the optimum found by BO,  $x^*$  the true optimum.

### 3.2 Deterministic model with 2 machines and 3 products (2M3P-D)

For this case, we know the optimum: both machines must be allocated 5 workers, the resulting average time in system (ATS) is 8 minutes. The experiment design has ten experiment runs with 15 iterations (5 initial random points, 10 suggested) and one replication (deterministic). The following table 3.1 shows the experiment run, the number of iterations needed to find the optimum, the worker at the machines, the average time in system (ATS) of the given parameters, and error, which is always zero for this model.

#### 3.2.1 SimPy

<i>Experiment run</i>	<i>Optimum found after <math>n</math> iterations</i>	<i>Machine 1</i>	<i>Machine 2</i>	<i>ATS</i>	<i>Error</i>
1	14	5	5	8	0
2	11	5	5	8	0
3	10	5	5	8	0
4	7	5	5	8	0
5	8	5	5	8	0
6	10	5	5	8	0
7	14	5	5	8	0
8	9	5	5	8	0
9	10	5	5	8	0
10	14	5	5	8	0

Table 3.1: Results of the 2M3P-D experiment in SimPy with 10 runs.

BO found the optimum 10 out of 10 times for the 2M3P-D model. The average number of iterations required until the optimum is found is 10.7. The reason for the difference in required iterations in a deterministic model are the random initial points. The sixth point is suggested based on the information BO got from the first 5 random points.

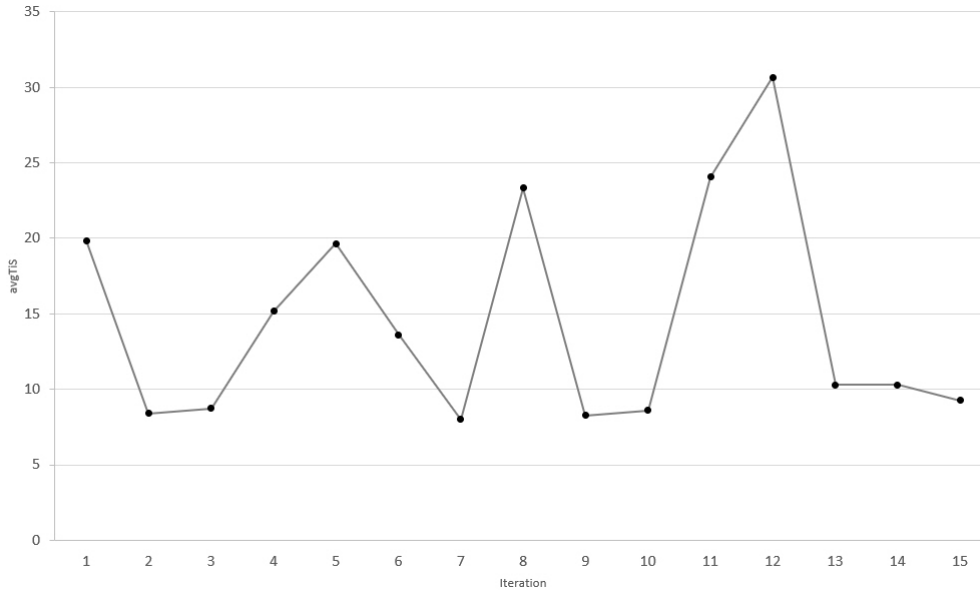


Figure 3.3: Evaluation values over 15 iterations of a single optimization run for the 2M3P-D model.

Figure 3.3 shows the evaluations of the experiment run 7. We can see that BO switches between exploration and exploitation. In this case the optimum was found at iteration 7.

### 3.2.2 Simio

To compare results between SimPy and Simio, we built the 2M3P-D model with both tools. Both models have the same setup and behave identical.

<i>Experiment run</i>	<i>Optimum found after <math>n</math> iterations</i>	<i>Machine 1</i>	<i>Machine 2</i>	<i>ATS</i>	<i>Error</i>
1	9	5	5	8	0
2	7	5	5	8	0
3	-	6	4	8.389	0.389
4	11	5	5	8	0
5	9	5	5	8	0
6	9	5	5	8	0
7	9	5	5	8	0
8	10	5	5	8	0
9	8	5	5	8	0
10	7	5	5	8	0

Table 3.2: Results of the 2M3P-D experiment in Simio with 10 runs.

The results are summarized in table 3.2. We can see that BO found the real optimum 9 out of 10 times. In run number 3, BO got close to the optimum (error = 0.389). The average number of iterations for the Simio experiment is 8.78.

### 3.3 Stochastic model with 5 machines and 5 products (5M5P-S)

For the 5M5P-S model, we skipped the SimPy implementation and built it only with Simio. The parameter range for each input value (machine capacity) was any value between 1 and 10. We put a constraint of 25 on the total maximum number of workers that can be allocated to machines. For this model, we ran one experiment with 100 iterations, starting again with 5 initial random points.

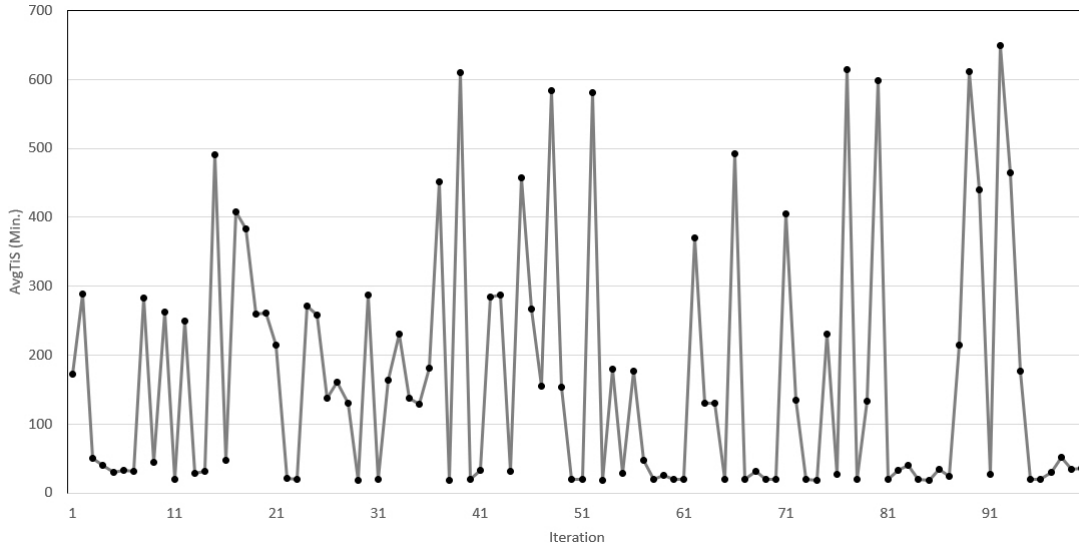


Figure 3.4: Evaluation values over 100 iterations of a single optimization run for the 5M5P-S Simio model.

Figure 3.4 shows the result of the experiment run with 100 iterations. We can see that BO again balances between exploration and exploitation to find the global optimum. Very early, it tries promising points with good responses values, but when the expected improvement gets too low, BO switches to exploration in different areas of the parameter space. This way, the risk to get stuck in a local minimum is reduced. In this case, the lowest value was found at iteration 38 with a ATS of 18.868 minutes.

## 4 Conclusion

### 4.1 Summary

In this paper, we have shown that BO finds the optimum of a simple (two dimensional) resource allocation problem fast requiring an average of 10.7 iterations (SimPy) and 8.78 iterations (Simio). Moreover, BO has shown to reliably find the optimum (SimPy 10 out of 10, Simio 9 out of 10). Furthermore, it has indicated good results for a similar five dimensional problem (possible minimum found after 38 iterations). The number of iterations in all experiments was acceptable. We conclude that the proof of concept of the application of BO to small discrete event simulation models was successful. We are at the beginning of our research, and many questions are still unanswered.

### 4.2 Further research questions

For our experiments in the course of this paper, the number of iterations BO required was acceptable. However, when the parameter space increases, so does the number of required iterations. Further research is needed to (1) analyze how the number of required iterations increases with larger parameter spaces, and (2) to find ways to keep the number of required iterations at a manageable level.

In our second experiment with the 5M5P-S model, we were confronted with the problem to evaluate BO's found optimum. Because we do not know the true optimum, we could rely on benchmarks to proof the correctness of the results. We plan to perform further experiments, in which we perform such benchmarks against popular simulation optimization tools such as OptQuest. This becomes especially relevant when we want to explore the application of BO to more realistic models we typically find in practice.

A simulation often contains constraints, which reduce the solution space. Most BO tools and frameworks are developed for hyperparameter tuning of machine learning algorithms, which do not have to deal with constraints. There are different ways how one can handle constraints with BO. For example, we could let the model respond with exceptionally bad values, so that BO learns not to evaluate points that lie outside of the parameter space. Another approach is to discard suggested values altogether, if they do not adhere to the constraints. GPyOpt employs the latter strategy, and we have seen performance go down as soon as complex constraints are part of the optimization problem. It is an open question to us which strategy to employ in the context of simulation optimization to maximize the performance of BO and keep performance up at the same time.

BO is originally designed to optimize functions with continuous parameters. However, in practice and in the context of simulation, we often encounter problems with discrete (integer, choice) parameters and mixed variants. Different approaches exist to address this challenge (e.g. rounding to the next integer), but all of them have their shortcomings. The best approach for the domain of simulation optimization is yet to be investigated.

In the described experiments, we applied Bayesian Optimization with a standard configuration as shipped by GPyOpt, without any optimizations. However, there are well known levers for optimization of the BO algorithm: (1) Choose a more appropriate acquisition function, which in our case was expected improvement (EI), (2) optimize the acquisition function's parameters, such as the balance between exploration and exploitation, and (3) analyze the fit of the surrogate model, which in our case were Gaussian processes, for the specific problem domain. For example, current research indicates that random forests might be a more suitable surrogate model for problems with discrete parameters [17].

## References

- [1] Michael C. Fu. *"Handbook of Simulation Optimization"*. Springer Publishing Company, Incorporated, 2014. ISBN: 1493913832, 9781493913831.
- [2] The Simpy authors. *SimPy: Discrete-Event Simulation for Python*. <https://simpy.readthedocs.io/en/latest/>. 2016.
- [3] The Simio authors. *Simio Simulation: Forward Thinking*. <https://www.simio.com/index.php>. 2019.
- [4] Abhijit Gosavi. "Simulation-Based Optimization: An Overview". In: *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Boston, MA: Springer US, 2015, pp. 29–35.
- [5] Hamed Jalali and Inneke Nieuwenhuyse. "Simulation optimization in inventory replenishment: A classification". In: *IIE Transactions* 47 (Apr. 2015), pp. 00–00.



- [6] Syusuke Sano et al. “Application of Bayesian Optimization for Pharmaceutical Product Development”. In: *Journal of Pharmaceutical Innovation* (Mar. 2019).
- [7] Daniel Golovin, Greg Kochanski, and John Elliot Karro. “Black Box Optimization via a Bayesian-Optimized Genetic Algorithm”. In: *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. To be submitted to Opt2017 Optimization for Machine Learning, at NIPS 2017. 2017.
- [8] Eric Brochu, Vlad M. Cora, and Nando de Freitas. “A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning”. In: *arXiv e-prints* (Dec. 2010).
- [9] Peter I. Frazier. “A Tutorial on Bayesian Optimization”. In: *arXiv e-prints* (July 2018).
- [10] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. “Practical Bayesian Optimization of Machine Learning Algorithms”. In: *arXiv e-prints* (June 2012).
- [11] Nacim Belkhir. “Per Instance Algorithm Configuration for Continuous Black Box Optimization”. PhD thesis. Nov. 2017.
- [12] James Bergstra et al. “Algorithms for hyper-parameter optimization”. In: *Advances in Neural Information Processing Systems* (Jan. 2011), pp. 2546–2554.
- [13] Ruben Martinez-Cantin. “BayesOpt: A Bayesian Optimization Library for Nonlinear Optimization, Experimental Design and Bandits”. In: *Journal of Machine Learning Research* 15 (2014), pp. 3915–3919.
- [14] B. Shahriari et al. “Taking the Human Out of the Loop: A Review of Bayesian Optimization”. In: *Proceedings of the IEEE* 104.1 (Jan. 2016), pp. 148–175.
- [15] Greg Kochanski et al. “Bayesian Optimization for a Better Dessert”. In: Dec. 2017.
- [16] The GPyOpt authors. *GPyOpt: A Bayesian Optimization framework in Python*. <http://github.com/SheffieldML/GPyOpt>. 2016.
- [17] A. Candelieri, R. Perego, and F. Archetti. “Bayesian optimization of pump operations in water distribution systems”. In: *Journal of Global Optimization* 71.1 (May 2018), pp. 213–235.

## Author Biographies

**PHILIPP ZMIJEWSKI** is a research assistant at the University of Applied Sciences in Osnabrueck. In context of his Master thesis, he is doing research in the field of simulation optimization, with focus on applying Bayesian Optimization.

**NICOLAS MESETH** is a professor for Information Systems at the University of Applied Sciences in Osnabrueck. He received his PhD from the University of Osnabrueck in 2011. His current research focuses on applications of machine learning algorithms to real world problems, mostly from the food manufacturing industry. A particular research interest is the combination of simulation and optimization with the use of machine learning algorithms.